SMART Surgery: Combining Finetuning Methods for Multitask BERT

Stanford CS224N Default Project

Ethan Foster
Department of Computer Science
Stanford University
epf@stanford.edu

- Mentor: Rohan Taori
- External Collaborators: None
- Sharing project: No

Abstract

We implement basic functionality for the minBERT model for multitask performance on three downstream tasks: sentiment analysis on the Stanford Sentiment Treebank [1] (Socher et al., 2013), paraphrase detection using the Quora dataset[2], and semantic textual similarity scoring on the STS SemEval Benchmark dataset (Agirre et al., 2013). We then extend the minBERT model with several modifications, focusing on different finetuning techniques including SMART loss as described in Jiang et al. (2019) and gradient surgery as described in Yu et al. (2020). We also augment the STS dataset to double our training data. We find that these finetuning methods can effectively be combined to improve performance, with our models that use SMART loss in combination with one or both of the other techniques performing the best on our dev datasets.

1 Introduction

Training machine learning models to perform various tasks has been a widely adopted practice in computer science research. Natural language provides us with a means to express a large variety of different tasks. Thus, using the power of natural language processing, we are able to create and train models that take in and output natural language in order to perform tasks. With the adoption of multi-head attention, these models are also able to "focus" on several different parts of the input sentences, making them a natural choice for a model that can perform multiple different tasks.

We use Bidirection Encoder Representations from Transformers ("BERT") model to produce a single vector representation from an input sentence (Devlin et al., 2019). Our model uses multi-head attention and feeds BERT's outputted vector into separate output layers for each task (sentiment analysis, paraphrase detection, and textual similarity scoring). We then train the model using our datasets to maximize performance on all three tasks together. This work is an exploration in fine-tuning specifically–adjusting the full breadth of the BERT model parameters and task output layers. We focus on three main approaches in our fine-tuning attempts: SMART loss that introduces a strong regularizer to prevent overfitting and aggressive updating proposed by Jiang et al. (2019), gradient surgery to prevent conflicting gradients between different tasks proposed by Yu et al. (2020), and augmenting the STS SemEval Benchmark dataset to double the training examples. These approaches are used both individually, and in combination to determine their effects on the model and levels of success.

[1] https://nlp.stanford.edu/sentiment/treebank.html
[2] https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs

2 Related Work

There has been plenty of work in fine-tuning approaches to increase performance in different downstream tasks. Our approach is inspired by the work of Jiang et al. (2019) and their proposed SMART loss for fine-tuning pretrained large language models. To combat aggressive over-fitting, they employ smoothness-inducing adversarial regularization and Bregman proximal point optimization. In their work, Jiang et al. found that SMART loss improves performance in all five tested downstream tasks when multitask training, and offered further increased benefits when combined with other fine-tuning methods.

Gradient surgery proposed by Yu et al. (2020) offered another approach to fine-tuning. In their paper, Yu et al. identified a source of conflict while fine-tuning for multitask learning. Named the "tragic triad", the combination of conflicting gradients, dominating gradients (that occur when one gradient has a much larger magnitude than the other), and high curvature can cause stalls in learning and sub-optimal performance. Gradient surgery combats this by identifying cases of the "tragic triad" and in those cases, projecting the gradient on one task onto the normal plane of the conflicting gradient. Yu et al. also found that their proposed gradient surgery increased performance in several downstream tasks and is able to be combined with other fine-tuning approaches for further gains.

Henderson et al. (2017) propose a different approach to fine-tuning: multiple negatives ranking loss learning. This approach provides a feed-forward approach to score consistency between two terms. It does this by taking a set of K pairs where each pair belong together, and terms from different pairs do not. It then minimizes the distance between terms in each pair while maximizing distance between terms in different pairs. This approach could be used where each pair consists of two similar sentences. Fine-tuning would then minimize the distance between similar sentences and maximize the distance between differing sentences, potentially improving the embeddings in our BERT model.

Based on these related works, our approach aims to combine the efforts of Jiang et al. (2019) and Yu et al. (2020) since both works provide a fine-tuning approach that has shown results for multi-task learning specifically, and are able to be combined. We include the multiple negatives ranking loss method of Henderson et al. (2017) as an alternative or additional approach that could be taken, but one that we will not utilize in our project due to time constraints.

3 Approach

3.1 Baseline minBERT

Using the provided scaffolding and default project handout, we first implement minBERT functionality by implementing the multi-head attention layer of the transformer, the normalization and forward functions of the BertLayer, and retrieval of the BERT embeddings. We then implement the sentiment classifier to perform our first task, sentiment analysis. We encode our BERT encoded sentences and after obtaining the pooled representation, we classify the sentence using dropout followed by a linear layer. Then, we implement the step() funciton of the Adam optimizer using provided scaffolding and based on the description in Loshchilov and Hutter (2017). Using our implemented minBERT functionality and our Adam optimizer using a cross entropy loss function, we finetuned the model on the sentiment analysis task to achieve a baseline performance of 0.515, recorded in our results table.

In order to complete the other two downstream tasks, we add two more dropout layers followed by linear layers. These layers are designed for the paraphrase detection and semantic textual similarity scoring tasks, so they must take in two sentences. Thus, we concatenate the embeddings from the two sentences before feeding them into these layers, which output a single logit. Binary cross entropy loss is used for the paraphrase detection task, and mean squared error is used for semantic textual similarity scoring. To train the model on all three tasks at once, we batch sample from each dataset to obtain a batch for each task, use our output layers to obtain our logits, calculate the loss for each task, and sum them together before updating our model with our Adam optimizer. The performance of this baseline finetuning model is recorded in the table under results.

3.2 SMART Loss

Given the complexity of our BERT model and the limited supply of our training examples, overfitting to our training data will cause our model to generalize poorly to unseen examples, such as our testing

dataset. We implement SMART Loss as described in Jiang et al. (2019) to help prevent this aggressive overfitting while finetuning. This method has two main parts. First, smoothness-inducing adversarial regularization controls the high complexity of the model. The idea behind this step is to make the model resistant to slight variations or perturbations in the embeddings. It does so by optimizing the equation:

$$\min_{\theta} \mathcal{F}(\theta) = \mathcal{L}(\theta) + \lambda_s R_s(\theta) \tag{1}$$

where the loss function $\mathcal{L}(\theta)$ is:

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i; \theta), y_i) \tag{2}$$

and $\ell(\cdot;\cdot)$ is our chosen task loss function, and λ_s is a tuning parameter. $\mathcal{R}_s(\theta)$, our smoothness-inducing adversarial regularizer is defined to be:

$$\mathcal{R}_s(\theta) = \frac{1}{n} \sum_{i=1}^{n} \max_{\|\tilde{x}_i - x_i\|_p \leq \epsilon} \ell_s(f(\tilde{x}_i; \theta), f(x_i; \theta)), \tag{3}$$

and $\ell_s(\cdot;\cdot)$ is chosen to be symmetrized KL-divergence ($\ell_s(P, Q) = \mathcal{D}_{KL}(P\|Q) + \mathcal{D}_{KL}(P\|Q)$) for classification tasks, and squared error ($\ell_s(p, q) = (p - q)^2$) for regression tasks. Jiang et al. (2019) also propose that this optimization function can efficiently be solved using gradient ascent, which we implement.

Secondly, SMART loss involves Bregman proximal point optimization in order to penalize large updates at each step. It does so using the equation:

$$\theta_{t+1} =_{\theta} \mathcal{F}(\theta) + \mu \mathcal{D}_{\text{Breg}}(\theta, \theta_t) \tag{4}$$

where $\mathcal{D}_{\text{Breg}}(\cdot;\cdot)$, the Bregman divergence is defined to be:

$$\mathcal{D}_{\text{Breg}}(\theta, \theta_t) = \frac{1}{n} \sum_{i=1}^{n} \ell_s(f(x_i; \theta), f(x_i; \theta_t)) \tag{5}$$

The second term in Equation 4 regularizes the updates to prevent them from being too large. We implemented SMART loss on our own, following the algorithm described in Jiang et al. (2019). We finetune the BERT model on all three downstream tasks together, summing the SMART loss from each task before updating our model, similarly to our baseline model. We call this model SMART BERT and record the performance of SMART BERT in the results table.

3.3 Gradient Surgery

Another consideration in finetuning our BERT model was that task gradients may conflict with one another. Conflicting task gradients are not necessarily bad on their own, and summing task gradients can lead to good performance under certain circumstances, as noted by Yu et al. (2020). However, as Yu et al. also note, there are circumstances in which adding the gradients can lead to significantly poorer performance as well, when compared to individual task finetuning. Yu et al. name these specific circumstances the "tragic triad", and they are composed of three simultaneous conditions: 1) Gradients from multiple tasks conflict with one another. They define conflicting as when the cosine of the angle between the gradients is less than zero. 2) The difference in magnitude of the conflicting gradients is large, causing one to dominate over the other. They define the gradient magnitude similarity between gradients $\mathbf{g}_i$ and $\mathbf{g}_j$ to be $\mathbf{\Phi}(\mathbf{g}_i, \mathbf{g}_j) = \frac{2\|\mathbf{g}_i\|_2 \|\mathbf{g}_j\|_2}{\|\mathbf{g}_i\|_2^2 \|\mathbf{g}_j\|_2^2}$. This value goes to zero as the difference in magnitude of conflicting gradients increases. 3) There is a high curvature within the multitask optimization landscape. Yu et al. define the multitask curvature to be $\mathbf{H}(\mathcal{L}; \theta, \theta') = \int_0^1 \delta\mathcal{L}(\theta)^T \delta^2 \mathcal{L}(\theta + a(\theta' - \theta)) \delta\mathcal{L}(\theta) da$. When $\mathbf{H}(\mathcal{L}; \theta, \theta')$ is large for model parameters θ and θ' between the current and next iteration, then the optimization landscape is characterized as having high curvature. Yu et al. show that the presence of these three conditions leads to the Adam optimizer stalling at sub-optimal plateaus providing empirical evidence of the tragic triad and its effects in multitask finetuning.

Yu et al. (2020) propose a method to avoid the tragic triad by altering the gradients to prevent conflict between them. Their method looks at each pair of gradients, and if they conflict, they project the first gradient onto the normal plane of the second gradient. This process is illustrated in Figure 1.

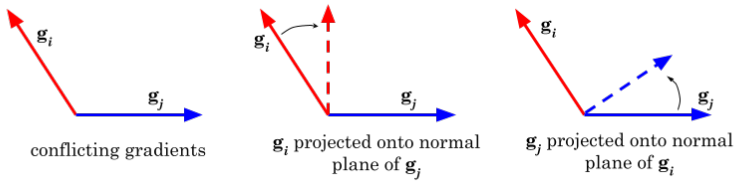

Figure 1: Conflicting gradients like the example above will have gradient surgery performed. The gradients that conflict will each be projected onto the normal plane of the other.

Yu et al. check for conflicts by calculating cosine similarity and classify negative cosine similarities as indicative of conflict. We implemented this finetuning method on our own, following the algorithm provided by Yu et al. (2020) in their original paper. To finetune our BERT model using gradient surgery, we sample from each of our three tasks. After calculating the loss for each task, we take each task's gradients, perform gradient surgery to resolve conflicts, and then sum our updated gradients to update our model. We call this model PCGrad BERT, taking the naming convention from Yu et al. (2020) where it stands for projecting conflicting gradients. We record the performance of PCGrad BERT in the results table.

3.4 Augmented STS Data

In order to maximize the use of our limited dataset, especially our smallest dataset, the STS SemEval Benchmark dataset, we looked for ways to augment the data and provide additional training examples. Looking at the semantic textual similarity scoring task, we realized that it seemed symmetric in nature. That is, if sentence A and sentence B have a similarity score of 3.6, then sentence B and sentence A would also presumably have a similarity score of 3.6.

Building off this intuition, we believed that we could double our dataset if we were to re-use each training example, providing the sentences in the reversed order. We implement this data augmentation step on our own, modifying the provided code to handle the SemEval dataset. With our changes, the training examples for the semantic textual similarity scoring task doubled, and we also hoped to capture the symmetric quality of this task in our finetuned embeddings. We record the performance of Augmented finetune in the results table. We finetune the model the same way as our baseline minBERT, however we can increase the batch size for our semantic textual similarity scoring task. We call this model stsAug BERT (short for semantic textual similarity augmented BERT), and record its performance in the results table.

3.5 Combined Approaches

All of the above approaches were chosen in part because they are able to be combined with other finetuning methods. As such, we were also able to combine each of them with one another to see how that would improve or otherwise affect the performance on the three downstream tasks. We combine them in every combination: SMART+PCGrad, SMART+stsAug, PCGrad+stsAug, and SMART+PCGrad+stsAug and finetune the original BERT model using them. The same hyperparameters, which are described in our Experimental Details, are used for each approach, although the batch size for the SemEval Benchmark dataset is doubled when stsAug is used. The performance from each combination is recorded in the results table.

4 Experiments

4.1 Data

We use three main datasets for our three downstream tasks, and one additional dataset as a benchmark during our implementation of minBERT. Our first task is sentiment analysis, and is trained and evaluated on the Stanford Sentiment Treebank, provided by Socher et al. (2013). This dataset is made up of 11,855 sentences taken from movie reviews and contains 215,154 unique phrases. Of this dataset, 8,544 sentences are used in the training set, 1,101 are used in the dev set, and 2,210 are

used for testing. Each sentence is labeled negative, somewhat negative, neutral, somewhat positive, or positive using an integer 0 to 4. For this first classification task, our model takes in the embeddings of the single sentence and outputs the predicted probability distribution for the 5 categories. During our implementation of minBERT, we also use the CFIMDB dataset for our benchmark tests. This dataset consists of 2,434 movie reviews that are highly polar, and each has a binary label of negative or positive. 1,701 of these reviews are used in the training set, 245 are used in our dev set, and 488 are used in our test set. Our model similarly takes in the embeddings and outputs the predicted probability distribution for the binary classification. The CFIMDB dataset is used during the implementation of minBERT only, and is not used for the finetuning extensions.

Our second task is paraphrase detection and is trained and evaluated using the Quora dataset provided to us in the starter code. We are provided 141,506 examples to train our model with, 20,215 examples for our dev set, and then a testing set of 40,431 examples. Each example consists of a pair of questions with a binary label to indicate if the questions are paraphrases of each other. For our paraphrase detection class, our model takes in a concatenation of the two questions' embeddings, and outputs a single logit that is the predicted probability that the questions in the input pair are paraphrases of one another.

Our last task is semantic textual similarity scoring, and we use the SemEval STS Benchmark dataset to train and evaluate this task. This dataset contains 8,628 pairs of sentences with labels that indicate how similar each pair of sentences is. The similarity is given as a continuous value on the scale of 0 (unrelated) to 5 (equivalent meaning). Of this dataset, 6,041 pairs are used for training, 864 are used in the dev set, and 1,726 are used in testing. For this task, our model takes in a concatenation of the two sentences' embeddings, and it outputs a single continuous value between 0 and 5, indicated the predicted similarity value between the sentences.

4.2 Evaluation method

To evaluate performance in downstream tasks for each model, we use percentage accuracy for sentiment analysis and paraphrase detection, and we use Pearson correlation for semantic textual similarity, following the evaluation method used by Agirre et al. (2013). The scores reported in the results table are calculated using our dev sets for all models, with three models chosen to receive scores calculated from our additional testing set. The datasets used for evaluation are labeled in the table. The accuracy for sentiment analysis and paraphrase detection are calculated by simply dividing the number of correct predictions outputted by the number of examples used in evaluation. For the semantic textual similarity task, the Pearson correlation coefficient is calculated between the true similarity values and the predicted similarity values for the evaluation data. These three evaluation metrics are also combined into an overall score upon submitting to a class-wide dev leaderboard. This is done by first converting the Pearson correlation from a [-1, 1] range to a [0, 1] range, and then averaging the three scores of the tasks. For each model design, we keep the model with the highest overall score.

4.3 Experimental details

We run all of our training with the same hyperparameters of learning rate, dropout rates, and epochs. We use a learning rate of 1e-5, dropout rates of 10% in within the BERT layers and 30% in the individual task heads, and 10 epochs. Our implemented ADAMW optimizer uses a learning rate of 1e-3, beta values of 0.9 and 0.999, and an epsilon of 1e-6.

For SMART loss, we set our hyperparameters to $\lambda = 5$, $\epsilon = 1e-4$, $\sigma = 1e-4$, $\mu = 1$, and $\eta = 1e-2$, following the range of appropriate hyperparameter values from Jiang et al. (2019), and after experimenting with several values and selecting the combination with the highest performance. These same hyperparameters are used for all models that use SMART loss.

Our batch sizes for the three downstream tasks are 2 for sentiment analysis, 20 for paraphrase detection, and 2 for textual semantic similarity scoring. When training our models with stsAug, we set our batch size to 4 for textual semantic similarity, since our training data has doubled.

4.4 Results

We submitted all our trained models to the dev leaderboard to determine our scores on the dev dataset. Additionally, we submitted our three highest performing models from the dev dataset to a separate test leaderboard to determine our scores on the test dataset. We record all of our models' performances in the table below. The highest performance in each task is bolded. There are separate columns for our dev performance and test performance, labeled for clarity. All models are finetuned for multitask performance except the first baseline model which is finetuned on only sentiment analysis.

Model	sst dev	para dev	sts dev	sst test	para test	sst test
Baseline finetuned single-task	0.515					
Baseline pretrained	0.404	0.550	0.223			
Baseline finetune	0.511	0.750	0.345			
SMART finetune	0.502	0.759	0.371			
PCGRAD finetune	0.500	0.760	0.352			
stsAug finetune	0.489	0.768	0.352			
SMART+PCGrad finetune	0.510	0.758	0.360	**0.525**	0.762	**0.336**
SMART+stsAug finetune	0.524	**0.774**	0.343	0.512	**0.779**	0.331
PCGrad+stsAug finetune	0.499	0.757	0.360			
SMART+PCGrad+stsAug finetune	**0.525**	0.752	**0.375**	0.520	0.755	0.334

We found that the highest dev performances were from models that used SMART loss. The model that combined all three approaches (SMART+PCGrad+stsAug finetune) had the highest scores for sentiment analysis and semantic textual similarity scoring, while the model that combined SMART loss with augmented sts data (SMART+stsAug finetune) had the highest score for paraphrase detection.

The models that combined multiple finetuning approaches generally outperformed the baselines and those that used only a single finetuning approach, which was unsurprising to us. One surprising finding was that SMART finetune outperformed SMART+stsAug finetune despite the increased training data in the latter model. Another surprising finding was that while the SMART+PCGrad+stsAug finetune model achieved the best performance for sentiment analysis and semantic textual similarity in the dev dataset, it did not perform the best in the test dataset, with SMART+PCGrad finetune outperforming it in all three tasks.

5 Analysis

We look at the outputs of our models against the true values in the datasets to identify cases where our model performs poorly and try to understand why this may be. Looking at the inaccuracies from our sentiment analysis task in the dev dataset, we find that our model generally predicts the sentiment within 1 level of the true sentiment. That is, most of the errors come from instances such as predicting "positive" when the true category is "somewhat positive", or predicting "somewhat negative" when the true category is "neutral". Many of these mistakes can be reasonably understood, especially when taking into account the subjectivity of rating the sentiment of reviews. We find ourselves agreeing with the "incorrect" predictions over the "true" labels in some cases, such as rating " It's a lovely film with lovely performances by Buy and Accorsi" as "positive" rather than "somewhat positive." However, there are other cases where our model's errors can not be explained by personal opinion and the subjectivity of the task. One example is with the review: "It moves quickly, adroitly, and without fuss; it doesn't give you time to reflect on the inanity–and the Cold War datedness–of its premise" which was predicted as "negative" but was labeled "neutral" as its true label. This review addresses some potentially negative aspects of the film, such as its "inanity" and "Cold War datedness," but ultimately partly disregards these negative aspects. Our models may have focuses too much on these negative aspects, resulting in too negative of a prediction overall. The nuances of the review may have been lost in the sentence embedding.

Looking at our failed predictions in the paraphrase detection task in the dev dataset, there are a few patterns that we notice. One pattern is that our models consistently mislabel a paraphrased question pair when one question includes a pronoun and the other does not. For example, "What makes one angry?" and "What is the one thing that makes you most angry?" was mis-predicted as not a

paraphrase of one another. The latter sentence contains the pronoun "you", while the prior sentence does not. There are several instances of mislabeling sentences such as these, and this could be due to how the model has learned to treat pronouns. Perhaps the model has learned, from finetuning on the semantic textual similarity task for instance, that the presence of pronouns, especially differing pronouns, can change the similarity of a sentence. This could lead to the finetuned embeddings reflecting very different values depending on the presence of pronouns, resulting in poor performance for paraphrase detection in these instances.

We also look at the failed predictions in the semantic textual similarity scoring task in the dev dataset. Here, our errors are a little harder to interpret as there seems to be pretty consistent over-scoring as well as under-scoring in many cases. However, one case of over-scoring seems to occurs when there are multiple words that overlap in the two sentences. For example, the sentences "UN chief welcomes peaceful presidential elections in Guinea" and "UN chief condemns attack against peacekeepers in Mali" consistently had a predicted similarity of 3 or higher, signifying a high level of similarity. This may be due to the overlap of the words "UN" and "chief" as well as the potentially similar embeddings for "peaceful" and "peacekeepers" as well as for "Guinea" and "Mali." Although in reality these two sentences have very different meanings, the model may have focused too much on the presence of these overlapping words, resulting in over-scoring the similarity.

One other pattern that we noticed, and were surprised by, was that the use of augmented SemEval STS Benchmark data did not consistently improve performance on the semantic textual similarity scoring task. Looking at our results table, we see that SMART finetune model scored a 0.371 for this task. However, introducing this augmented data in the SMART+stsAug finetune model resulted in a drop of score to 0.343, our lowest score in this task besides the Baseline pretrained. We believe that the introduction of this augmented data may have caused the model to overfit to the limited data it had. Simply swapping the order of the sentences may not have been enough to truly diversify the data more, resulting in poorer performance in the dev and test datasets.

6 Conclusion

In our project, we show how SMART loss, gradient surgery, and augmented data, used individually or in combination, can improve the performance of the multitask BERT model for the three downstream tasks: sentiment analysis, paraphrase detection, and semantic textual similarity scoring. Our best performing models all used SMART loss in combination with one or both of the other methods, indicating that these finetuning methods can be effectively combined with one another for increased performance. These finetuning methods aim to decrease the complexity of the model, reduce aggressive overfitting, prevent conflicting gradients, and increase the size of datasets. However, these approaches are not perfect, and while they do each increase overall performance, models trained with these approaches still exhibit overfitting and in some cases, decreased performance in certain individual tasks compared to the baseline finetuning without any extensions.

In the future, we could further explore adjusting hyperparameters for our current approaches to find an optimal use of SMART loss and gradient surgery. Another future route would be to implement the multiple negatives ranking loss learning proposed by Henderson et al. (2017) and mentioned in our related work section. This is another approach to finetuning that differs from our currently implemented methods and would further our exploration in finetuning the BERT model.

References

Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. 2013. *sem 2013 shared task: Semantic textual similarity. *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*.

Matthew Henderson, Rami Al-Rfou, Brian Strope, Yun-Hsuan Sung, László Lukács, Ruiqi Guo, Sanjiv Kumar, Balint Miklos, and Ray Kurzweil. 2017. Efficient natural language response suggestion for smart reply. *arXiv*.

Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2019. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. *arXiv preprint*.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. *Proceedings of the 2013 conference on empirical methods in natural language processing*.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020. Gradient surgery for multi-task learning. *Conference on Neural Information Processing Systems (NeurIPS)*.